# EXPOSING J2C INTERFACE PROPERTIES

## COPYRIGHT NOTICE

## BACKGROUND OF THE INVENTION

### Field of the Invention

The present invention relates to Web Services Invocation Framework (WSIF) operations and, in particular, to exposing J2C interface properties in such operations.

### Description of the Related Art

Historically, Enterprise Information/Integration Systems (EIS), such as legacy mainframe systems, and, more recently, commercial applications, have been at the center of modern information technology environments, providing critical data for enterprise operations. Companies have big investments in their Enterprise Information Systems. While many applications may be old, some may be new. The EIS provides the company with the quality of service required by the company. An example application, known as a “transactional” application, allows updates to be made to a bank account. Although these systems continue to be critically important to many businesses, these businesses may further rely on other applications. As these other applications are often web or wireless applications, there appears to be an increasing need to integrate web and wireless applications with legacy mainframe systems and existing commercial applications. Such integration requires the real-time exchange of information and services by a range of interested parties. The new applications, for which this integration is required, are increasingly being developed in the Java™ programming language.

As a required element of the Java™ 2 Enterprise Edition (J2EE), the Java Connector Architecture (J2C) provides a standardized means to integrate Java applications with EISs (Java is a Trademark of Sun Microsystems, Inc.) J2C defines a Common Client Interface (CCI). The CCI defines a standard client Application Programming Interface (API) for application components. The CCI enables application components and Enterprise Application Integration (EAI) frameworks to drive interactions across heterogeneous EISs using a common client API. Interfaces that are part of the CCI include "interactionSpec" and "connectionSpec"; these interfaces are further described hereinafter.

As mentioned hereinbefore, companies may have business requirements to enable web access to these services. To extend the previous example of a banking application, there may be a requirement to enable access to bank accounts over the Internet. The J2EE Connector architecture (J2C) provides an environment so that an EIS can provide a resource adapter that can plug in to any application server so that the application server can generically provide qualities of service to all resource adapters, and optionally, through the CCI, the resource adapter can implement the common client programming model for the enterprise application. Such J2C services may further be extended to be web services.

Integration of Java applications with EIS may be accomplished through the use of J2C. The Web Services Description Language (WSDL), with its extensions, allows the description of many different kinds of services: Web services, Java services, Enterprise Java Bean (EJB) services, Java Message Service (JMS) services, J2C services, etc. The Web Services Invocation Framework (WSIF - see http://ws.apache.org/wsif) provides a common way of invoking each of these services. The WSIF supports a simple Java™ API for invoking Web services. Use of the WSIF API allows clients to invoke services focusing on an abstract service description, that is, the portion of WSDL that covers port types, operations and message exchanges without referring to real protocols.

As mentioned hereinbefore, WSDL is extensible to allow the description of many types of services other than web services. With WSIF, any of these types of services may be invoked in a common fashion. However, WSIF does not expose certain J2C properties, thus constraining certain applications.

## SUMMARY OF THE INVENTION

interactionSpec and connectionSpec properties are exposed as data in WSIF operations, thus WSIF support for J2C is made functionally more complete. Advantageously, exposing J2C interactionSpec and connectionSpec properties as data in a WSIF operation allows the connectionSpec and interactionSpec properties to be set dynamically on input and the interactionSpec properties to be retrieved dynamically on output.

In accordance with an aspect of the present invention there is provided a method of improving Web Services Invocation Framework support for Java 2 Enterprise Edition Java Connector Architecture comprising exposing properties of a given interface as data.

In accordance with another aspect of the present invention there is provided a method of a performing Web Services Invocation Framework operation. The method includes receiving an input message that includes a plurality of parts, determining whether any of the plurality of parts are instances of a property of a given interface and, if a given part of the plurality of parts is determined to be an instance of the property of the given interface, setting a value from the given part into the given interface, thereby exposing a property of the given interface as data. In other aspects of the present invention, a resource adapter is provided in an application sever, the resource adapter for performing this method and a computer readable medium is provided to allow a general purpose computer to perform this method.

Other aspects and features of the present invention will become apparent to those of ordinary skill in the art upon review of the following description of specific embodiments of the invention in conjunction with the accompanying figures.

## BRIEF DESCRIPTION OF THE DRAWINGS

In the figures which illustrate example embodiments of this invention:

FIG. 1 illustrates an exemplary environment for implementation of aspects of the present invention;

FIG. 2 illustrates steps of a method carried out by a resource adapter in the environment of FIG. 1 according to an embodiment of the present invention;

FIGS. 3A-3R illustrate exemplary code for implementing WSIF Operation aspects of the present invention;

FIGS. 4A-4Q illustrate exemplary code for implementing WSIF Provider External Call Interface aspects of the present invention;

FIGS. 5A-5D illustrate exemplary code for implementing interactionSpec Property exposure aspects of the present invention;

FIGS. 6A-6D illustrate exemplary code for implementing ConnectionSpec Property exposure aspects of the present invention;

FIGS. 7A-7H illustrate exemplary code for implementing WSIF Provider Extension aspects of the present invention;

FIGS. 8A-8O illustrate exemplary code for implementing Streamable Message aspects of the present invention;

FIGS. 9A-9D illustrate exemplary code for implementing WSIF Binding Operation aspects of the present invention;

FIGS. 10A-10E illustrate an exemplary customer port type definition implementing aspects of the present invention;

FIGS. 11A-11I illustrate an exemplary binding definition implementing aspects of the present invention;

FIG. 12 illustrates an exemplary services definition implementing aspects of the present invention; and

FIGS. 13A-13H illustrate exemplary client code implementing aspects of the present invention.

## DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. 1 illustrates an exemplary environment for implementation of aspects of the present invention. In particular, a web client 102 is illustrated in communication with a web server 104. It should be understood that this connection will typically occur over a digital communications network. Where an end user at the web client is interested in a service provided by an EIS 110, such an interest can be indicated to the web server 104 and, in response to receiving such an indication, the web server 104 may contact an application server 106. It is then the task of the application server 106 to interact with the EIS 110 as directed by the end user.

As mentioned hereinbefore, J2C provides an environment so that the EIS 110 can provide a resource adapter 108 that can plug in to the application server 106 so that the application server 106 can generically provide the qualities of services available from the resource adapter 108 illustrated and other EIS-specific resource adapters (not shown).

The resource adapter 108 may be loaded with methods exemplary of this invention from a software medium 112 which could be a disk, a tape, a chip or a random access memory containing a file downloaded from a remote source.

Much of the following description of interactionSpec and connectionSpec and respective properties is drawn from the Java™ 2 Platform, Enterprise Edition, v 1.3 API Specification (hereby incorporated herein by reference).

The interactionSpec property holds properties for driving an interaction with an EIS instance. The CCI specification defines a set of standard properties for an interactionSpec. An interactionSpec implementation is not required to support a standard property if that property does not apply to its underlying EIS. The interactionSpec implementation class must provide getter and setter methods for each of its supported properties. The getter and setter methods convention should be based on the Java Beans design pattern. The standard properties are as follows:

FunctionName: name of an EIS function
InteractionVerb: mode of interaction with an EIS instance
ExecutionTimeout: the number of milliseconds an Interaction will wait for an EIS to execute the specified function
FetchSize
FetchDirection
MaxFieldSize
ResultSetType
ResultSetConcurrency

The last five standard properties are used to give hints to an Interaction instance about the ResultSet requirements.

A CCI implementation can provide additional properties beyond that described in the interactionSpec interface. Note that the format and type of the additional properties is specific to an EIS and is outside the scope of the CCI specification.

ConnectionSpec is used by an application component to pass connection request-specific properties to an interface for getting connection to an EIS instance. The CCI specification defines a set of standard properties for a ConnectionSpec. The properties are defined either on a derived interface or an implementation class of an empty ConnectionSpec interface. In addition, a resource adapter may define additional properties specific to its underlying EIS. A resource adapter is a system-level software driver that is used by a Java application to connect to an EIS. The resource adapter plugs into an application server and provides connectivity between the EIS, the application server and the enterprise application. In simpler terms, a resource adapter is the software that allows an application to access functions in an EIS.

Among others, the following standard properties are defined by the CCI specification for ConnectionSpec:
UserName: name of the user establishing a connection to an EIS instance
Password: password for the user establishing a connection

interactionSpec and ConnectionSpec Properties generally need to be “exposed” to allow an application to set the values of the properties at execution time. Otherwise, the values of the properties would have to be set when the application is built.

Often the interactionSpec values may be preset. However, some scenarios require the ability to set a value or to obtain a value on output. For example, in “component managed signon”, there is a requirement of the ability to set the user name and password on the connectionSpec on input. In contrast, for “container managed signon”, a connectionSpec is not used.

ConnectionSpec properties are part of the port section of the WSDL. ConnectionSpec exposes any security information and connection parameters that are specific to the resource adapter. In the component managed signon case, an application component passes security information (example: username, password) through a ConnectionSpec instance. interactionSpec properties are part of the binding section of the WSDL.

It is known that J2C has a managed and a non-managed scenario. In the non-managed scenario the connectionSpec and interactionSpec properties are taken from an associated WSDL file. An interactionSpec object can also be exposed as a property on a proxy that is used to execute an EIS interaction.

In the managed scenario, some connectionSpec properties (UserName and Password) are exposed as an administered Java Authentication and Authorization Service (JAAS) Subject or as custom properties on an interface for getting connection to an EIS instance. interactionSpec properties are not exposed in the managed scenario.

Exposing connectionSpec properties is important for enabling component managed signon. Exposing interactionSpec properties is important because some EIS interactionSpec properties are important for the application to set or retrieve at runtime. Unfortunately, when working with WSIF J2C operations, the interactionSpec properties are part of the binding and are not exposed. A "binding" is a WSDL concept. A binding defines the concrete implementation of an abstract operation. The binding specifies the message format and protocol details of the abstract operation. In the context of a WSIF J2C operation, the binding is communicated to the resource adapter 108 in one or more messages.

To implement the exposure of the interactionSpec properties and the connectionSpec properties as data, the method typically implemented by the resource adapter 108 is altered as illustrated in FIG. 2.

An input message is received by the resource adapter 108 from the application server 106, causing the resource adapter 108 to perform the steps of the method 200 illustrated in FIG. 2. The updateInteractionSpec method provided by the EIS 110 is initially called (step 202) to update the interactionSpec using data from the input message. The method called updateInteractionSpec determines whether any of the parts in the input message are instances of the interactionSpecProperty. If a part is determined to be such an instance, updateInteractionSpec takes the value from the part and sets the value into the interactionSpec. It is then determined whether a connection is currently available (step 204). Where a connection is not available, the connection is considered to be “null”. If a connection is not currently available, it is determined whether any of the parts in the input message are instances of the connectionSpecProperty (step 206). If so, while creating a connection to the EIS 110 (step 207), values from each of such parts are set into the connectionSpec, which is then used when creating the connection. If no parts are instances of the connectionSpecProperty, a connection to the EIS 110 is created with no connectionSpec specified (step 208). In each of steps 207 and 208, the connection to the EIS 110 is created by calling a createConnection method. If a connection is determined (step 204) to be currently available, or once a connection has been created (step 207 or 208), an interaction (i.e., a javax.resource.cci.Interaction) is created from the connection (step 209). An interaction execute method is then invoked (step 210) with the EIS 110. The interaction is then closed (step 212). If the interaction execute method is an input only method, the method is complete. However, where the interaction execute method is a request response operation (determined in step 214), the interactionSpec is set into an output message (step 216) and the output message is updated (step 218) with specified interactionSpec properties.

Note that, in the port type section of the WSDL, the developer adds a part to the input message for each interactionSpec or connectionSpec property that is to be exposed as data. Parts are also added to the output message for each interactionSpec property that is to be exposed as data. In the binding section of the WSDL for the input and output sections of the operation, the developer then specifies how these added parts map to connectionSpec or interactionSpec properties. This mapping is then used at runtime.

As will be apparent to a person skilled in the art, the portion of the method 200 illustrated in FIG. 2 that is typically implemented by the resource adapter 108 is represented by step 209, step 210 and step 212.

Exemplary code operable to implement the method of FIG. 2 is illustrated in FIGS. 3A-3R as WSIFOperation_JCA.java.

The method called updateInteractionSpec that is used in WSIFOperation_JCA.java (see FIGS. 3D and 3F) and called at step 202 of FIG. 2, is provided as part of exemplary code illustrated in FIGS. 4A-4Q, called WSIFProvider_ECI.java (in particular, see FIG. 4G). The ECI acronym relates to the External Call Interface of the resource adapter 108. The method called updateInteractionSpec determines whether any of the parts in the input message are instances of the interactionSpecProperty. If a part is determined to be such an instance, updateInteractionSpec takes the value from the part and sets the value into the interactionSpec. The method called updateInteractionSpec calls ECIInteractionSpecProperty, which is illustrated in FIG. 5.

The method called createConnection that is used in WSIFOperation_JCA.java (see FIGS. 3E and 3G) and called at step 206 of FIG. 2 is also provided as part of WSIFProvider_ECI.java (see FIG. 4L). The method called createConnection determines whether any of the parts in the input message are instances of the connectionSpecProperty. If a part is determined to be such an instance, createConnection takes the value from the part, sets the value into the connectionSpec and uses the connectionSpec when creating the connection. If no parts are instances of the connectionSpecProperty, a connection is created with no connectionSpec specified. The method called createConnection calls ECIConnectionSpecProperty, which is illustrated in FIG. 6.

Additionally, a method called updateOutputMessage that is used in WSIFOperation_JCA.java (see FIG. 3E) and called at step 218 of FIG. 2 is provided as part of WSIFProvider_ECI.java (see FIG. 4K).

A known interface called WSIFProviderJCAExtensions has been updated as shown in FIG. 7. to include the updateInteractionSpec, updateOutputMessage and createConnection methods.

WSIFMessage_JCAStreamable, which is illustrated in FIG. 8, keeps the input interactionSpec and connectionSpec properties, which were provided as parts in the input message, from being sent as data to the EIS 110. WSIFMessage_JCAStreamable also populates the parts of the output message as appropriate from the interactionSpec.

WSIFBindingOperation_JCAProperty, which is illustrated in FIG. 9, is an interface of the methods used to get/set partName and either connectionSpec property name or interactionSpec property name.

Consider the use of an embodiment of the present invention in the following example. A message called "getCustomerRequest" is defined in Customer.wsdl (see FIGS. 10A-10E, in particular FIG. 10E). The message definition includes identification of particular parts, including "userid", "password" and "functionName". An operation called "getCustomer" is also defined in Customer.wsdl. The getCustomer message receives the specific getCustomerRequest message as input and issues a getCustomerResponse message as output.

The getCustomer operation is further defined in CustomerCICSECIBinding.wsdl (see FIGS. 11A-11I). By this further definition, the user name part and the password part are each defined as a ConnectionSpecProperty and the functionName part is defined as an InteractionSpecProperty. Thus, the user name part and the password part may be exposed on the ConnectionSpec as data and the functionName part may be exposed on the interactionSpec as data.

The definitions of exemplary definition files Customer.wsdl and CustomerCICSECIBinding.wsdl are imported into the exemplary definition file called CustomerCICSECIService.wsdl (FIG. 12). The acronym CICS refers to the "Customer Information Control System", which is a family of application servers and connectors that provides online transaction management and connectivity for applications.

Exemplary program code, CustomerProxy.java, is illustrated in FIGS 13A-13H. Notably, CustomerProxy.java references the exemplary definition file called CustomerCICSECIService.wsdl (see FIG. 13D) and, by doing so, thus references exemplary definition files Customer.wsdl and CustomerCICSECIBinding.wsdl. The reference to these exemplary definition files allows the exemplary program code to make reference to the getCustomer operation and the getCustomerRequest and getCustomerResponse messages (see FIG. 13C).

As will be apparent to a person skilled in the art, exposing interactionSpec and ConnectionSpec properties as data allows the size of the commarea of a CICS external call interface to be specified, providing a performance enhancement (where a commarea is a "communications area", which defines the input and output for a program). Additionally, the exposing allows a user name and a password to be passed by a resource adapter when establishing a connection, thus supporting component managed signon.

Other modifications will be apparent to those skilled in the art and, therefore, the invention is defined in the claims.